

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МО ЭВМ**

**ОТЧЕТ**  
**по практической работе №4**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Бинарное дерево**

Студент гр. 8304

Бочаров Ф.Д.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2019

### **Задание.**

#### **Вариант №5-В**

Задано бинарное дерево *b* типа *BT* с произвольным типом элементов. Используя очередь и операции над ней, напечатать все элементы дерева *b* по уровням: сначала из корня дерева, затем (слева направо) из узлов, сыновних по отношению к корню, затем (также слева направо) из узлов, сыновних по отношению к этим узлам, и т. д.

### **Цель работы.**

Научиться программировать бинарные деревья.

### **Описание алгоритма.**

После считывания применяется алгоритм обхода бинарного дерева в ширину, с использованием очереди.

Поиск в ширину работает путём последовательного просмотра отдельных уровней графа, начиная с узла-источника. Рассмотрим все рёбра выходящие из узла. Если очередной узел является целевым узлом, то поиск завершается; в противном случае узел добавляется в очередь. После того, как будут проверены все рёбра, выходящие из узла, из очереди извлекается следующий узел, и процесс повторяется.

### **Описание функций программы:**

```
1. BinTree(int n)
```

Конструктор класса `BinTree` принимает аргументом размер начального массива.

*N* – размер исходного массива.

```
2. bool make_bin_tree(int& counter, std::string& str)
```

Функция предназначена для создания бинарного дерева и проверки строки, введенной пользователем.

Counter – счётчик символов входной строки

Str – строка, введенная пользователем

```
3. int left(BinTree::elem* current_elem)
```

Функция, возвращающая индекс левого корня в массиве.

Elem – элемент для проверки

```
4. int right(BinTree::elem* current_elem)
```

Функция, возвращающая индекс правого корня в массиве.

Elem – элемент для проверки

```
5. void queue_output(elem* root, std::queue<elem*>& queue)
```

Рекурсивная функция для вывода элементов дерева путем обхода в ширину.

Root – элемент, который будет считаться корнем обрабатываемого дерева.

Queue – очередь для хранения элементов, в необходимом для вывода порядке.

```
6. void resize_array()
```

Функция предназначена для выделения дополнительной памяти, если выделенной изначально оказалось недостаточно.

```
7. void set_elem_value(T value, int index)
```

Функция предназначена для занесения значения в элемент массива по заданному индексу.

T –

Index –

```
8. void set_left_child_index(int current_index, int child_index = -1)
```

Функция предназначена для установки индекса левого ребёнка данного элемента массива.

Current\_index – индекс элемента, в которых производится

запись Child\_index – индекс элемента-ребенка

```
9. void set_right_child_index(int current_index, int child_index = -1)
```

Функция предназначена для установки индекса правого ребёнка данного элемента массива.

Current\_index – индекс элемента, в которых производится запись  
Child\_index – индекс элемента-ребенка

```
10. elem* get_root()
```

Функция возвращает корень дерева, которое сохранено в массиве

### Выводы.

Для решения данной задачи нецелесообразно было вообще использовать бинарное дерево, тем более его реализацию на векторе, которая не имеет практически никаких преимуществ над реализацией с помощью динамической памяти.

## Протокол

### Тестирование:

Входные данные	Выходные данные
(a(b(c)(d))(g))	a b g c d
(v(c)(d))	v c d
(a(c(t)(w))(c))	a c c t w
(13(2#(24))(3))	132324
wfawawf53 5	Unexpected input format
a(b(c))awfweges36j	Unexpected input format
_/<awfawfa	Unexpected input format

## Исходный код

### Main.cpp

```
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <string>
#include <array>
#include <queue>

template <typename T>
class BinTree{
public:

    struct elem{
        T value;
        int left_child_index;
        int right_child_index;
    };

    explicit BinTree(int n){
        MAX_SIZE = n;
        array = new elem*[n];
        for(int i = 0; i < n; i++){
            array[i] = new elem;
        }
    }

    void set_elem_value(T value, int index) {
        array[index]->value = value;
    }

    void set_left_child_index(int current_index, int child_index = -1) {
        array[current_index]->left_child_index = child_index;
    }

    void set_right_child_index(int current_index, int child_index = -1) {
        array[current_index]->right_child_index = child_index;
    }

    elem* get_root() {
        return array[0];
    }

    bool make_bin_tree(int& counter, std::string& str){

        int local_index = size;

        if(str[counter] != '(') {
            std::cout << "Unexpected input format" << std::endl;
            return false;
        }
        std::string current_substring;
        while((str[++counter] != '(') && (str[counter] != ')') &&
(str[counter] != '#') && (counter != str.size()))
            current_substring += str[counter];
        if((current_substring.empty()) && (str[counter] == '(')) {
            std::cout << "Unexpected input format" << std::endl;
            return false;
        }
    }
```

```

        set_elem_value(current_substring, local_index);

        if(str[counter] == ')') {
            set_left_child_index(local_index);
            set_right_child_index(local_index);
            ++counter;
            return true;
        }
        if(str[counter] == '#' && str[counter+1] != '(') {
            std::cout << "Unexpected symbol in the input stream(something
after '#' symbol)" << std::endl;
            return false;
        }
        else if(str[counter] == '#') {
            set_left_child_index(local_index);
            ++counter;
        }
        else if(str[counter] == '('){
            if(size + 1 == MAX_SIZE){
                resize_array();
            }
            set_left_child_index(local_index, ++size);
            if(!make_bin_tree(counter, str))
                return false;
        }
    }

    if(str[counter] == ')') {
        set_right_child_index(local_index);
        ++counter;
        return true;
    }
    else if(str[counter] == '('){
        if(size + 1 == MAX_SIZE){
            resize_array();
        }
        set_right_child_index(local_index, ++size);

        if(!make_bin_tree(counter, str))
            return false;
    }
    else {
        std::cout << "Unexpected input format" << std::endl;
        return false;
    }
    ++counter;
    if(str[counter] == ' ') {
        std::cout << "Unexpected input format" << std::endl;
        return false;
    }
    return true;
}

int left(BinTree::elem* current_elem){
    return current_elem->left_child_index;
}

int right(BinTree::elem* current_elem){
    return current_elem->right_child_index;
}

void queue_output(elem* root, std::queue<elem*>& queue){

```

```

        std::cout << root->value << ' ';

        if(root->left_child_index != -1)
            queue.push(array[left(root)]);
        if(root->right_child_index != -1)
            queue.push(array[right(root)]);

        if(!queue.empty()) {
            elem* tmp = queue.front();
            queue.pop();
            queue_output(tmp, queue);
        }
    }

    void resize_array(){
        elem** extra_array = new elem*[MAX_SIZE*2];
        for(int i = 0; i <= size; i++){
            extra_array[i] = array[i];
        }

        MAX_SIZE = MAX_SIZE*2;
        for(int i = size + 1; i < MAX_SIZE; i++)
            extra_array[i] = new elem;
        delete[] array;
        array = extra_array;
    }

    ~BinTree(){
        for(int i = 0; i < MAX_SIZE; i++) {
            delete(array[i]);
        }
        delete[] array;
    }
private:
    int size = 0;
    int MAX_SIZE = 0;
    elem** array;
};

int main(int argc, char* argv[]) {

    std::string str;
    std::string check_str = " ";
    int counter = 0, test_counter = 0;

    std::queue<BinTree<std::string>::elem*> queue;

    if (argc > 1) {
        std::ifstream in(argv[1]);
        if (!in.is_open()) {
            std::cout << "This file can't be open" << std::endl;
            return 1;
        }
        while (std::getline(in, str)) {
            std::cout << std::endl << "Test #" << ++test_counter << " "
<< str << std::endl;
            size_t find_cnt = str.find_first_not_of(check_str, 0);
            str.erase(0, find_cnt);

```

```

        BinTree<std::string> tree(20);
        if(tree.make_bin_tree(counter, str)) {
            tree.queue_output(tree.get_root(), queue);
        }
        counter = 0;
    }
    return 0;
}
else {
    BinTree<std::string> tree(20);
    std::cout << "Enter binary tree" << std::endl;
    std::getline(std::cin, str);
    size_t find_cnt = str.find_first_not_of(check_str, 0);
    str.erase(0, find_cnt);
    if(!tree.make_bin_tree(counter, str))
        return 1;
    tree.queue_output(tree.get_root(), queue);
}

while(!queue.empty()) {
    delete(queue.front());
    queue.pop();
}
return 0;
}

```